Introduction

Javascript is a programming language that is widely used for web development. It was created in 1995 and has since become an essential part of creating interactive and dynamic web pages. In this 2-hour session, we will cover the fundamentals of Javascript, including data types, variables, operators, control structures, functions, and arrays. By the end of this session, you will have a solid understanding of Javascript and be able to write basic Javascript code.

Data Types and Variables

Javascript has several data types, including numbers, strings, booleans, null, and undefined. Numbers are used for mathematical operations, strings for text, booleans for true/false values, null for empty values, and undefined for variables that have not been assigned a value.

Variables are used to store data and assign values to them. In Javascript, variables can be declared using the keywords `let`, `var`, or `const`. `let` and `var` are used for mutable variables, while `const` is used for immutable variables. Once a variable is declared, its value can be changed by re-assigning it.

Here is an example of declaring a variable and performing basic operations with it:

```
let x = 5;
let y = 7;
let sum = x + y;

console.log("The sum of x and y is " + sum);
```

Operators and Control Structures

Operators are symbols or keywords used to perform operations on values. Javascript has several types of operators, including arithmetic, comparison, and logical operators. Arithmetic operators perform mathematical calculations such as addition, subtraction, multiplication, and division. Comparison operators compare two values and return a boolean value (true or false) based on the comparison

result. Logical operators are used to combine two or more boolean expressions and return a boolean value based on the logical result.

Control structures are used to control the flow of program execution. Javascript has several types of control structures, including if/else statements, switch statements, and loops. If/else statements are used to execute code based on a certain condition. Switch statements are used to compare a value with multiple cases and execute code based on the matched case. Loops are used to execute a block of code multiple times until a certain condition is met.

Here is an example of using operators and control structures in Javascript:

```
let num1 = 5;
let num2 = 7;
let sum = num1 + num2;

if (sum > 10) {
  console.log("The sum is greater than 10.");
} else {
  console.log("The sum is less than or equal to 10.");
}
```

Functions

Functions are reusable blocks of code that perform a specific task. They can be called multiple times with different arguments to produce different results. Functions can have parameters, which are variables used to receive values passed to the function, and a return statement, which is used to return a value to the caller.

Here is an example of declaring and calling a function in Javascript:

```
function addNumbers(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = addNumbers(3, 4);
console.log("The result is " + result);
```

Arrays

Arrays are used to store collections of data in Javascript. They can hold multiple values of different data types. Arrays can be initialized using brackets and can be accessed using the index of the value.

Here is an example of using arrays in Javascript:

```
let fruits = ["apple", "banana", "orange"];
console.log(fruits[1]);
```

Commonly used methods on Arrays

Arrays are a fundamental data type in Javascript and are widely used in web development. Javascript provides several built-in methods to work with arrays. Here are some commonly used methods on arrays in Javascript:

1. `push()` - Adds one or more elements to the end of an array.
2. `pop()` - Removes the last element from an array and returns it.
3. `shift()` - Removes the first element from an array and returns it.
4. `unshift()` - Adds one or more elements to the beginning of an array.
5. `slice()` - Returns a new array that contains a portion of an existing array.
6. `splice()` - Changes the contents of an array by adding or removing elements.
7. `concat()` - Joins two or more arrays and returns a new array.
8. `indexOf()` - Returns the index of the first occurrence of a specified element in an array.
9. `join()` - Joins all elements of an array into a string.
10. `reverse()` - Reverses the order of the elements in an array.

Here is an example of using some of these array methods in Javascript:

```
let fruits = ["apple", "banana", "orange"];
// add an element to the end of the array
fruits.push("pear");

// remove the first element from the array
fruits.shift();

// slice the first two elements from the array
let slicedFruits = fruits.slice(0, 2);
```

```
// add two elements to the beginning of the array
fruits.unshift("grape", "pineapple");

// remove the second element from the array and replace it with "kiwi"
fruits.splice(1, 1, "kiwi");

console.log(fruits); // ["grape", "kiwi", "orange", "pear"]
console.log(slicedFruits); // ["grape", "kiwi"]
```

Introduction to higher order functions

Higher-order functions are functions that take one or more functions as arguments or return a function as their result. Higher-order functions are a powerful tool in functional programming, as they enable the creation of more modular and reusable code.

Here is an example of using a higher-order function in Javascript:

```
function higherOrderFunction(func) {
  console.log("Before function call");
  func();
  console.log("After function call");
}

function myFunction() {
  console.log("Inside myFunction");
}

higherOrderFunction(myFunction);
```

In this example, `higherOrderFunction` is a higher-order function that takes a function `func` as an argument and calls it inside the function body.

Object declaration

Objects are a fundamental data type in Javascript and are used to represent real-world entities. An object is a collection of properties, where each property has a key and a value. The key is a string that identifies the property, and the value can be any Javascript data type, including other objects.

Here is an example of declaring an object in Javascript:

```
let person = {
  name: "John",
  age: 30,
  gender: "male",
  address: {
    street: "123 Main St",
    city: "New York",
    state: "NY",
    zip: "10001"
  }
};

console.log(person.name); // "John"
console.log(person.address.city); // "New York"
```

Accessing properties of objects

Properties of an object can be accessed using the dot notation or the bracket notation. In the dot notation, the property is accessed using the object name followed by a dot and the property name. In the bracket notation, the property is accessed using the object name followed by square brackets and the property name as a string.

Here is an example of accessing object properties in Javascript using both notations:

```
let person = {
  name: "John",
  age: 30,
```

```
  gender: "male",
  address: {
    street: "123 Main St",
    city: "New York",
    state: "NY",
    zip: "10001"
  }
};

console.log(person.name); // "John"
console.log(person["age"]); // 30
console.log(person.address["city"]); // "New York"
```

Object methods and their use case

Objects in Javascript can also have methods, which are functions that are stored as object properties. Methods can be used to perform operations on the object or to provide functionality related to the object.

Here is an example of declaring an object with a method in Javascript:

```
let person = {
  name: "John",
  age: 30,
  gender: "male",
  address: {
    street: "123 Main St",
    city: "New York",
    state: "NY",
    zip: "10001"
  },
  sayHello: function() {
    console.log("Hello, my name is " + this.name);
  }
};

person.sayHello(); // "Hello, my name is John"
```

In this example, `sayHello` is a method of the `person` object that prints a message to the console using the `name` property of the object.

Introduction to Dom

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM is an object-oriented representation of the web page, which can be modified using Javascript.

Dom tree and traversal

The DOM tree is a hierarchical structure that represents the HTML elements of a web page. Each HTML element is a node in the tree, and the relationships between elements are represented as parent-child relationships.

Here is an example of a simple DOM tree:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1>Hello World</h1>
  <p>This is a paragraph</p>
</body>
</html>
```

In this example, the `html` element is the root of the DOM tree. It has two child elements, the `head` element and the `body` element. The `body` element has two child elements, the `h1` element and the `p` element.

Traversing the DOM tree means moving from one element to another element in the tree. This can be done using several DOM traversal methods, such as `getElementById`, `getElementsByTagName`, `getElementsByClassName`, and `querySelector`.

Here is an example of using the `getElementById` method to retrieve an element from the DOM tree:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1 id="title">Hello World</h1>
  <p>This is a paragraph</p>
  <script>
    let title = document.getElementById("title");
    console.log(title.innerText); // "Hello World"
  </script>
</body>
</html>
```

In this example, the `getElementById` method is used to retrieve the `h1` element with an `id` of "title" from the DOM tree. The `innerText` property of the element is then logged to the console.

Reading DOM elements

DOM elements can be read using various properties and methods. Some of the commonly used properties to read DOM elements are `innerText`, `textContent`, `innerHTML`, `value`, `src`, `href`, `alt`, etc.

Here is an example of reading the `innerText` property of an element using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1>Hello World</h1>
  <p>This is a paragraph</p>
  <script>
    let heading = document.querySelector("h1");
    console.log(heading.innerText); // "Hello World"
  </script>
```

```
</body>
</html>
```

In this example, the querySelector method is used to retrieve the first h1 element from the DOM tree. The innerText property of the element is then logged to the console.

Dom methods

DOM methods are used to modify the structure, style, and content of the web page. Some of the commonly used DOM methods include createElement, appendChild, removeChild, setAttribute, addEventListener, classList, etc.

Here is an example of creating a new element and adding it to the DOM tree using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <div id="container"></div>
  <script>
    let container = document.getElementById("container");
    let paragraph = document.createElement("p");
    paragraph.innerText = "This is a new paragraph";
    container.appendChild(paragraph);
  </script>
</body>
</html>
```

In this example, the getElementById method is used to retrieve the div element with an id of "container" from the DOM tree. A new p element is then created using the createElement method, and its innerText property is set. The new element is added to the container element using the appendChild method.

Manipulating/Accessing the page content

DOM methods and properties can be used to manipulate and access the content of the web page. For example, the `innerText`, `textContent`, and `innerHTML` properties can be used to set or retrieve the content of an element. The `setAttribute` method can be used to set the value of an attribute, such as `src`, `href`, or `class`.

Here is an example of setting the content and attribute of an element using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <img id="image" src="" alt="">
  <script>
    let image = document.getElementById("image");
    image.setAttribute("src", "https://example.com/image.jpg");
    image.setAttribute("alt", "An example image");
  </script>
</body>
</html>
```

In this example, the `getElementById` method is used to retrieve the `img` element from the DOM tree. The `setAttribute` method is used to set the `src` and `alt` attributes of the element.

innerText, innerContent and innerHTML

The `innerText`, `textContent`, and `innerHTML` properties are used to access or set the content of an element. The `innerText` property returns the text content of an element, while the `textContent` property returns the text content of an element and its descendants, including any whitespace. The `innerHTML` property returns the HTML content of an element, including any child elements.

Here is an example of using the `innerHTML` property to set the content of an element using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <div id="container"></div>
  <script>
    let container = document.getElementById("container");
    container.innerHTML = "<p>This is a new paragraph</p>";
  </script>
</body>
</html>
```

In this example, the `getElementById` method is used to retrieve the `div` element with an `id` of "container" from the DOM tree. The `innerHTML` property is then used to set the content of the element to a new `p` element.

Dynamically creating content to the page

Javascript can be used to dynamically create and add content to a web page. The `createElement` method is used to create new elements, which can be modified and added to the DOM tree using other methods, such as `appendChild`.

Here is an example of dynamically creating a new element using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <div id="container"></div>
  <script>
    let container = document.getElementById("container");
    let paragraph = document.createElement("p");
    paragraph.innerText = "This is a new paragraph";
    container.appendChild(paragraph);
  </script>
</body>
```

```
</html>
```

In this example, the `getElementById` method is used to retrieve the `div` element with an `id` of "container" from the DOM tree. A new `p` element is then created using the `createElement` method, and its `innerText` property is set. The new element is added to the `container` element using the `appendChild` method.

Add or remove elements from page

Elements can be added or removed from the DOM tree using methods such as `appendChild`, `removeChild`, `replaceChild`, and `insertBefore`. These methods allow for dynamic manipulation of the structure of the web page.

Here is an example of adding and removing elements from the DOM tree using Javascript:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <div id="container">
    <p>This is a paragraph</p>
  </div>
  <script>
    let container = document.getElementById("container");
    let paragraph = document.querySelector("p");
    let newParagraph = document.createElement("p");
    newParagraph.innerText = "This is a new paragraph";
    container.replaceChild(newParagraph, paragraph);
  </script>
</body>
</html>
```

In this example, the `getElementById` method is used to retrieve the `div` element with an `id` of "container" from the DOM tree. The `querySelector` method is used to retrieve the `p` element within the `div`. A new `p` element is then created using the

createElement method, and its innerText property is set. The replaceChild method is used to replace the original p element with the new one.

Array Destructuring and Spread operator

Array destructuring and the spread operator are useful features in Javascript that allow for easier manipulation of arrays. Array destructuring allows for the extraction of individual elements from an array, while the spread operator can be used to combine multiple arrays into a single array or to create a copy of an array.

Here is an example of using array destructuring and the spread operator in Javascript:

```
let numbers = [1, 2, 3, 4, 5];
let [first, second, ...rest] = numbers;
console.log(first); // 1
console.log(second); // 2
console.log(rest); // [3, 4, 5]

let newNumbers = [...numbers, 6, 7];
console.log(newNumbers); // [1, 2, 3, 4, 5, 6, 7]
```

In this example, the array numbers is defined with five elements. The first and second variables are assigned the first two elements of the array using array destructuring. The rest variable is assigned an array containing the remaining elements using the rest operator (...). The spread operator (...) is also used to create a new array called newNumbers that contains all of the elements of numbers plus two additional elements.

Iterators on arrays

Javascript provides several built-in methods for iterating over arrays, including forEach, map, filter, reduce, and find. These methods can be used to perform various operations on each element of an array, such as modifying the values or selecting specific elements based on certain conditions.

Here is an example of using the forEach method to iterate over an array:

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(element) {
```

```
  console.log(element);
});
```

In this example, the `forEach` method is used to iterate over the `numbers` array. A function is passed as an argument to `forEach`, which is executed for each element in the array. The function simply logs each element to the console.

Higher order functions

Higher order functions are functions that take one or more functions as arguments or return a function as their result. These functions are commonly used in Javascript to create more flexible and reusable code.

Here is an example of a higher order function in Javascript:

```
function multiplyBy(factor) {
  return function(number) {
    return number * factor;
  }
}

let multiplyByTwo = multiplyBy(2);
let multiplyByThree = multiplyBy(3);

console.log(multiplyByTwo(5)); // 10
console.log(multiplyByThree(5)); // 15
```

In this example, the `multiplyBy` function is defined to take a `factor` argument and return a new function that multiplies its argument by the `factor`. Two new functions, `multiplyByTwo` and `multiplyByThree`, are created by calling `multiplyBy` with arguments of 2 and 3, respectively. These new functions can then be called with a number argument to multiply it by their respective factors.

Map, filter and reduce

Map, filter, and reduce are higher order functions that are commonly used in Javascript to perform operations on arrays. The `map` function is used to transform each element of an array, the `filter` function is used to select certain elements based on a condition, and the `reduce` function is used to aggregate the elements of an array into a single value.

Here is an example of using the `map`, `filter`, and `reduce` functions in Javascript:

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(element) {
  return element * 2;
});
console.log(doubledNumbers); // [2, 4, 6, 8, 10]

let evenNumbers = numbers.filter(function(element) {
  return element % 2 === 0;
});
console.log(evenNumbers); // [2, 4]

let sum = numbers.reduce(function(total, element) {
  return total + element;
}, 0);
console.log(sum); // 15
```

In this example, the `map` function is used to double each element in the `numbers` array, producing a new array called `doubledNumbers`. The `filter` function is used to select only the even numbers from the `numbers` array, producing a new array called `evenNumbers`. The `reduce` function is used to calculate the sum of all elements in the `numbers` array by iteratively adding each element to a `total` variable, starting with an initial value of 0.

Sort, forEach, find etc

In addition to `map`, `filter`, and `reduce`, there are several other built-in methods available for working with arrays in Javascript. The `sort` method is used to sort the elements of an array in place. The `forEach` method is used to iterate over the elements of an array, similar to the `map` method but without returning a new array. The `find` method is used to find the first element in an array that satisfies a given condition.

Here is an example of using these methods in Javascript:

```
let numbers = [5, 3, 2, 4, 1];

numbers.sort();
console.log(numbers); // [1, 2, 3, 4, 5]
```

```
numbers.forEach(function(element) {
  console.log(element);
});

let firstEvenNumber = numbers.find(function(element) {
  return element % 2 === 0;
});
console.log(firstEvenNumber); // 2
```

In this example, the `sort` method is used to sort the `numbers` array in ascending order. The `forEach` method is used to log each element of the `numbers` array to the console. Finally, the `find` method is used to find the first even number in the `numbers` array, returning the value `2`.

By understanding and utilizing these commonly used methods on arrays, higher order functions, object declaration, accessing properties and methods of objects, the DOM and its traversal, as well as dynamic creation and manipulation of page content, developers can create powerful and interactive applications using Javascript.